# Computer Vision

## Day 5

2017, Tanta University
**Eng.**Sara Hussien

# Image Transformation

# Image Transformation

- In many cases it will be useful to transfer the image to another domain for better processing or applying specific operator or perform further analysis.
- From these
  1) Enhancement
  2) Restoration e.g. fixing known distortion
  3) Compression
  4) Analysis

# Frequency Domain:

- The transformation domain of interest is the Frequency Domain.

  *Note: transformation === mapping

  T: Spatial domain $\longrightarrow$ Frequency Domain

  $f(x,y) \longrightarrow F(v,u)$

- The transformations of interest in the context of this course are:-

  1) Fourier transform

  2) Discrete cosine transform

# Fourier Transform

1) Single Dimension and Continuous Function

$$F(u) = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi jux} dx$$

$$F(u) = \mathcal{F}(f(x))$$

$$f(x) = \mathcal{F}^{-1}\big(F(u)\big) = \int_{-\infty}^{\infty} F(U)\, e^{2\pi jUx}\, du$$

2) Two Dimensions Continuous function:

$$F(u,v) = \mathcal{F}\big(f(x,y)\big) = \iint_{-\infty}^{\infty} f(x,y) e^{-2\pi j(ux+vy)} dxdy$$

$$f(x,y) = \mathcal{F}^{-1}\big(F(U,V)\big) = \iint_{-\infty}^{\infty} F(U,V) e^{2\pi j(UX+VY)} dudv$$

# Fourier Transform

3) Single Discrete:

$$F(u) = \frac{1}{N} \sum_{N=0}^{N-1} f(x) e^{\frac{-2\pi ux}{N}}$$

Given that $f(x) = \{f(0), f(1), \dots, f(N-1)\}$
Or $x = 0,1,2, \dots \dots , N-1$

4) Two Dimension Discrete:

$$f(x,y)$$

$x$=0,1,… ,$N$−1

$y$=0,1,… ,$M$−1

# Fourier Transform

4) Two Dimension Discrete:

$$F(U,V) = \mathcal{F}\big(f(x,y)\big) = \frac{1}{NM}\sum_{Y=0}^{M-1}\sum_{X=0}^{N-1} f(x,y)e^{-2\pi j(\frac{UX}{N}+\frac{VY}{M})}$$

$$f(x,y) = \mathcal{F}^{-1}\big(F(U,V)\big) = \sum_{V=0}^{M-1}\sum_{U=0}^{N-1} F(U,V)e^{2\pi j(\frac{UX}{N}+\frac{VY}{M})}$$

$f(x,y)$ *is a real domain matrix ,* $F(u,v)$ *is a complex domain*

Signal magnitude: (any element in the matrix have its own magnitude)

$$\|F(U,V)\| = \sqrt{R^2(U,V) + I^2(U,V)}$$

$$\text{the phase of an element is } \varphi\big(F(U,V)\big) = \tan^{-1}\frac{I(U,V)}{R(U,V)}$$

$$\|F(U,V)\|^2 = \text{Signal Power} = R^2(U,V) + I^2(U,V)$$

# Discrete Cosine Transform

- Transform of the discrete cosine is from
  Real Domain ⟶ Real Domain

Single Dimension:

$$c(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) \cos\left[\frac{(2x+1)\pi u}{2N}\right]$$

$$c(u) \rightarrow frequency\ Domain$$

Two Dimensional Discrete Cosine Transform:

$$c(u,v) = \alpha(u)\,\alpha(v) \sum_{y=0}^{M-1} \sum_{x=0}^{N-1} f(x,y) \cos\left(\frac{(2x+1)\pi u}{2N}\right) \cos\left(\frac{(2y+1)\pi v}{2M}\right)$$

$$f(x,y) = \sum_{v=0}^{M-1} \sum_{u=0}^{N-1} (u)\,\alpha(v) \cos\left(\frac{(2x+1)\pi u}{2N}\right) \cos\left(\frac{(2y+1)\pi v}{2M}\right)$$

# Image Enhancement in Frequency Domain

**Image Enhancement in Frequency Domain**

$f(x,y)$ —Enhancement→ $f'(x,y)$

Orignal Image

Enhanced Image

$\mathcal{F}(F(x,y)) \longrightarrow F(u,v)$ —Enhancement→ $F'(u,v)$

all in Frequency Domain

1) We convert the image to frequency domain
2) We enhance the image in frequency domain
3) Then convert it to $f(x,y)$

# Discrete Fourier Transform

- Fourier analysis is fundamentally a method for expressing a function as a

sum of periodic components, and for recovering the function from those components, (DFT) for discrete input function .

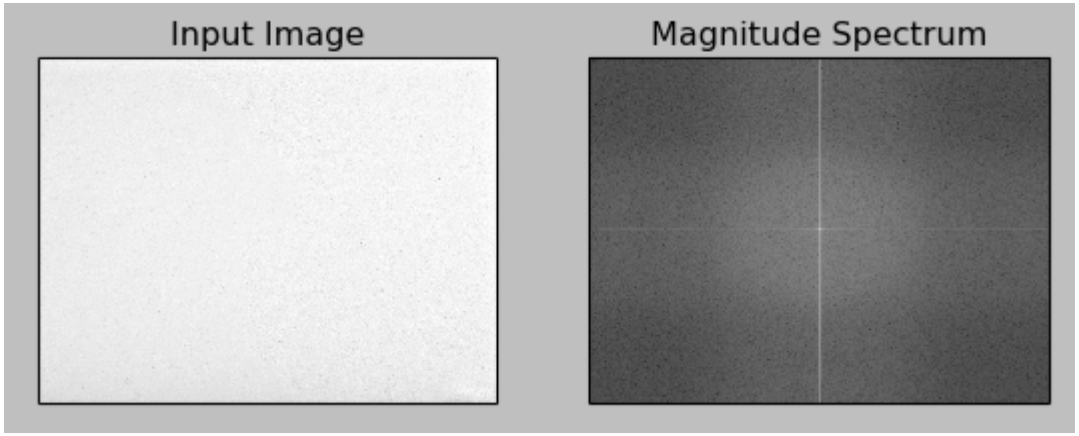
- The DFT has become a mainstay of numerical computing in part because of a **very fast algorithm for computing** it, called the Fast Fourier Transform (**FFT**).
- DFT has a great number of applications in digital signal processing, e.g., for filtering.
- the discretized input to the transform is referred to as a signal in time domain. The output is called a *spectrum* or *transform* and exists in the *frequency domain*.
- **Some terms**
  - **The least Frequency element (zero frequency component will be at top left corner)**
  - **The Highest Frequency element**

# Image Processing in the Fourier Domain

the Fourier Transform tells you what is happening in the image in terms the frequencies of those sinusoids:

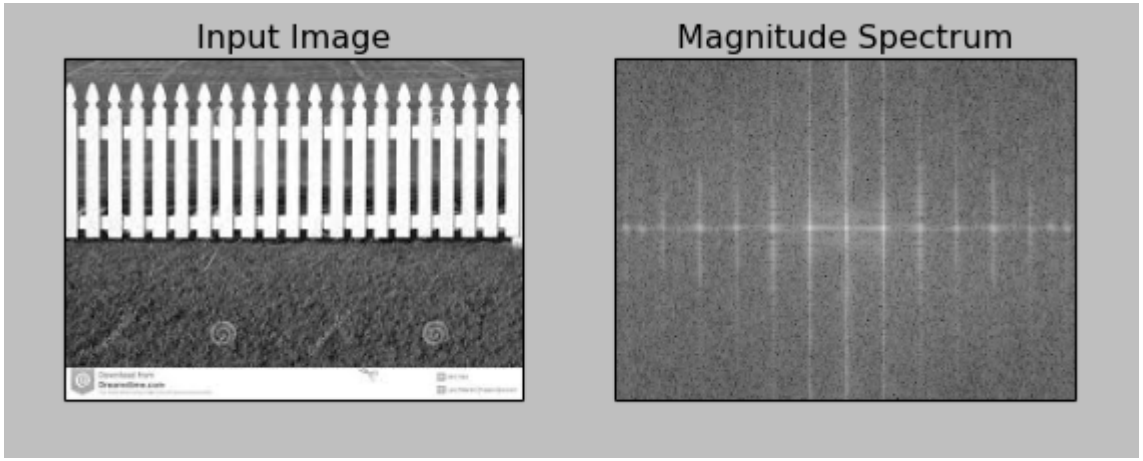
For example:

- **A picture of a plain wall** : the values of the pixels change very little as you go from left to right or from top to bottom. In the frequency domain that means that your image contains low frequencies, but no high frequencies.

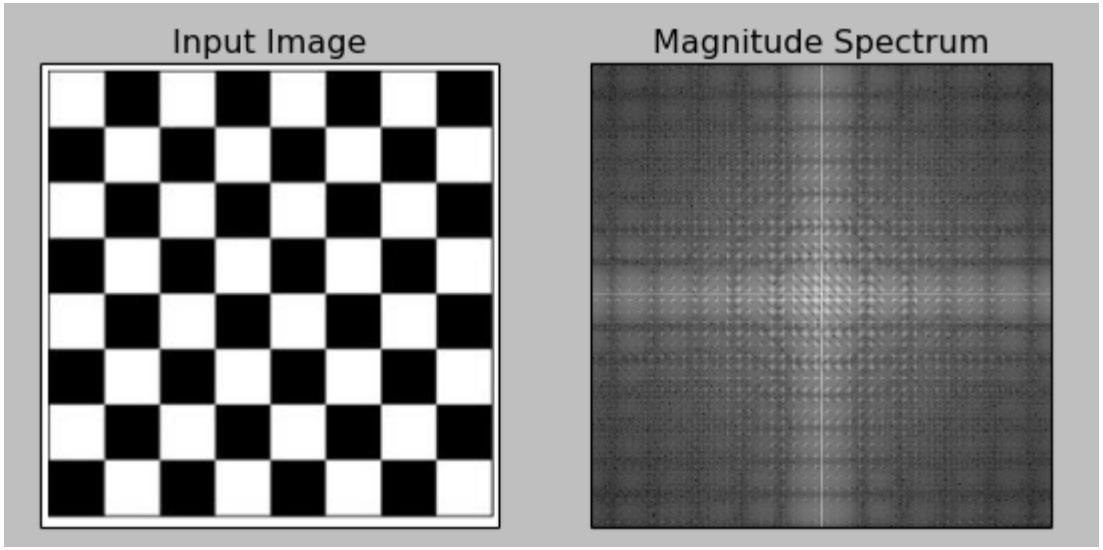


# Image Processing in the Fourier Domain

- **A picture of a picket Fence** : the values of the pixels change all the time as you go from left to right. So in Fourier domain you have high frequencies in the X direction, but not in the Y direction.

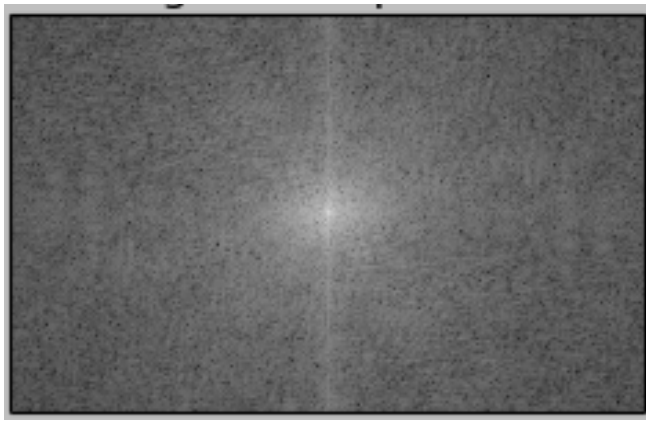



- **A picture of a checkerboard** : then the pixel values change a lot in both directions. Thus the Fourier transform of the image will have high frequencies in both X and Y.

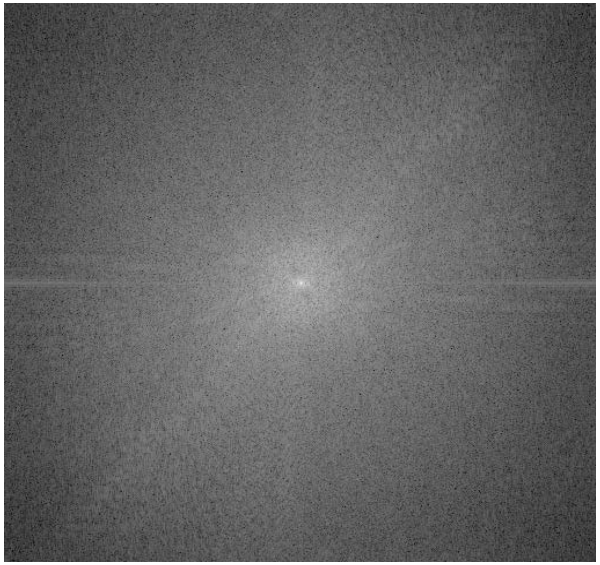


# Image Processing in the Fourier Domain

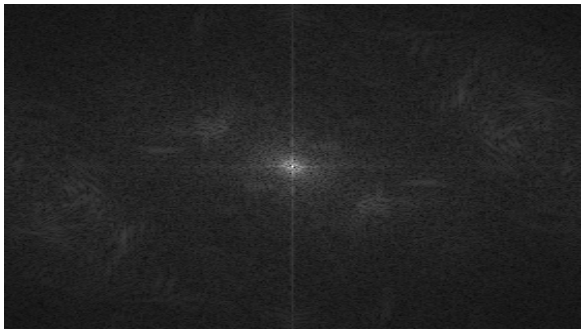
Magnitude of the FT

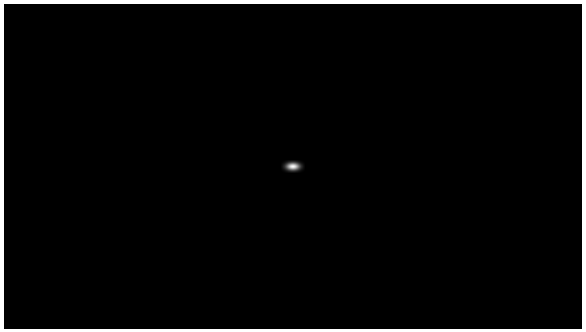
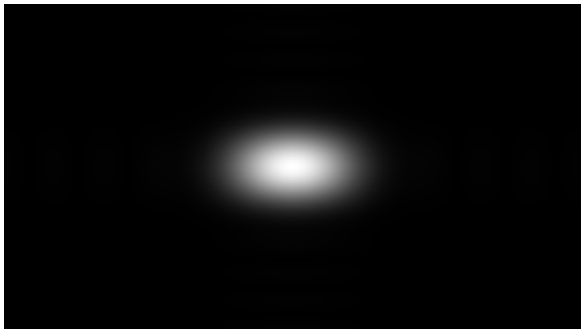


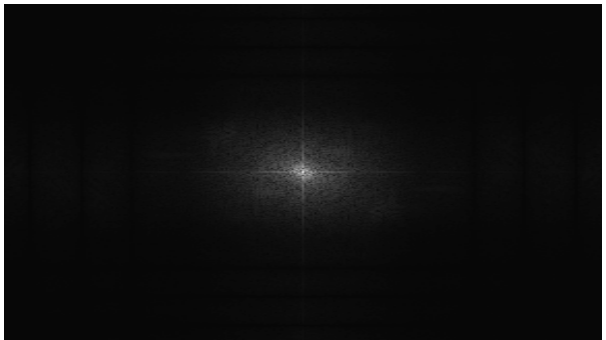
Magnitude of the FT



# Convolution is Multiplication in Fourier Domain

$f(x,y)$  $\Rightarrow$  $|F(s_x,s_y)|$

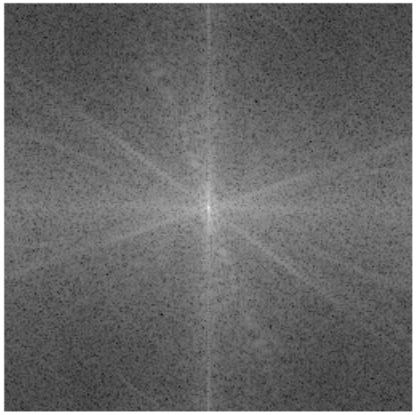
$h(x,y)$  $\Rightarrow$  $|H(s_x,s_y)|$

$g(x,y)$  $\Leftarrow$  $|G(s_x,s_y)|$

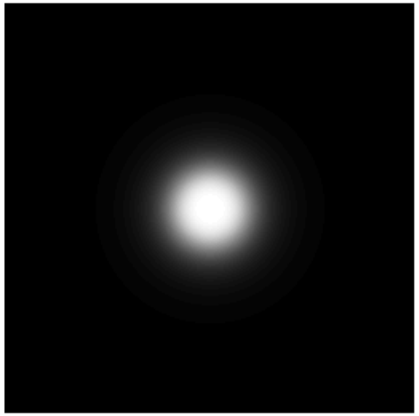
# Low-pass Filtering

Original image

FFT of original image



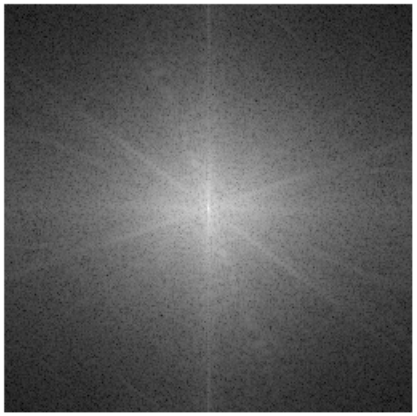
Low-pass filter



Let the low frequencies pass and eliminating the high frequencies.

Low-pass image

FFT of low-pass image

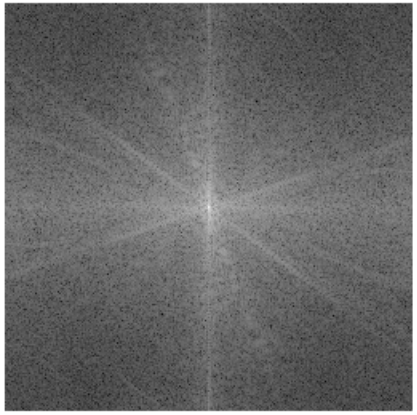


Generates image with overall shading, but not much detail

# High-pass Filtering

Original image

FFT of original image

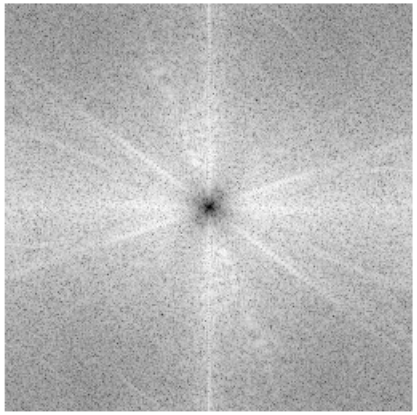


High–pass filter

Lets through the high frequencies (the detail), but eliminates the low frequencies (the overall shape). It acts like an edge enhancer.

High–pass image

FFT of high–pass image



# Discrete Fourier Transform in Numpy

- **np.fft.fft2() :** an FFT package that provides us the frequency transform which will be a complex array.
  - first argument is the input image in gray scale.
  - Second argument is the size of output array.(Optional)
    - If size > size of input image → input image is padded with zeros before calculation of FFT.
    - If size < size input image, input image will be cropped.
    - If no arguments passed, Output array size will be same as input.
- **np.fft.fftshift(A) :**shifts transforms and their frequencies to put the zero-frequency components in the middle, and np.fft.ifftshift(A) undoes that shift.
- np.fft.ifftshift() & np.fft.ifft2( )

# Discrete Fourier Transform in OpenCV

- **cv2.dft()** and **cv2.idft()** :
  - It returns the same result as previous, but with two channels. First channel will have the real part of the result and second channel will have the imaginary part of the result.
  - The input image should be converted to **np.float32** first.

```
dft = cv2.dft(np.float32(img),flags = cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)

magnitude_spectrum = 20*np.log(cv2.magnitude(dft_shift[:,:,0],dft_shift[:,:,1]))
```

```
>>> help(np.fft.fftshift)
Help on function fftshift in module numpy.fft.helper:

fftshift(x, axes=None)
    Shift the zero-frequency component to the center of the spectrum.

    This function swaps half-spaces for all axes listed (defaults to all).
    Note that ``y[0]`` is the Nyquist component only if ``len(x)`` is even.

    Parameters
    ----------
    x : array_like
        Input array.
    axes : int or shape tuple, optional
        Axes over which to shift.  Default is None, which shifts all axes.

    Returns
    -------
    y : ndarray

    Examples
    --------
    >>> freqs = np.fft.fftfreq(10, 0.1)
    >>> freqs
    array([ 0.,  1.,  2.,  3.,  4., -5., -4., -3., -2., -1.])
    >>> np.fft.fftshift(freqs)
    array([-5., -4., -3., -2., -1.,  0.,  1.,  2.,  3.,  4.])

    Shift the zero-frequency component only along the second axis:

    >>> freqs = np.fft.fftfreq(9, d=1./9).reshape(3, 3)
    >>> freqs
    array([[ 0.,  1.,  2.],
           [ 3.,  4., -4.],
           [-3., -2., -1.]])
    >>> np.fft.fftshift(freqs, axes=(1,))
    array([[ 2.,  0.,  1.],
           [-4.,  3.,  4.],
           [-1., -3., -2.]])
```